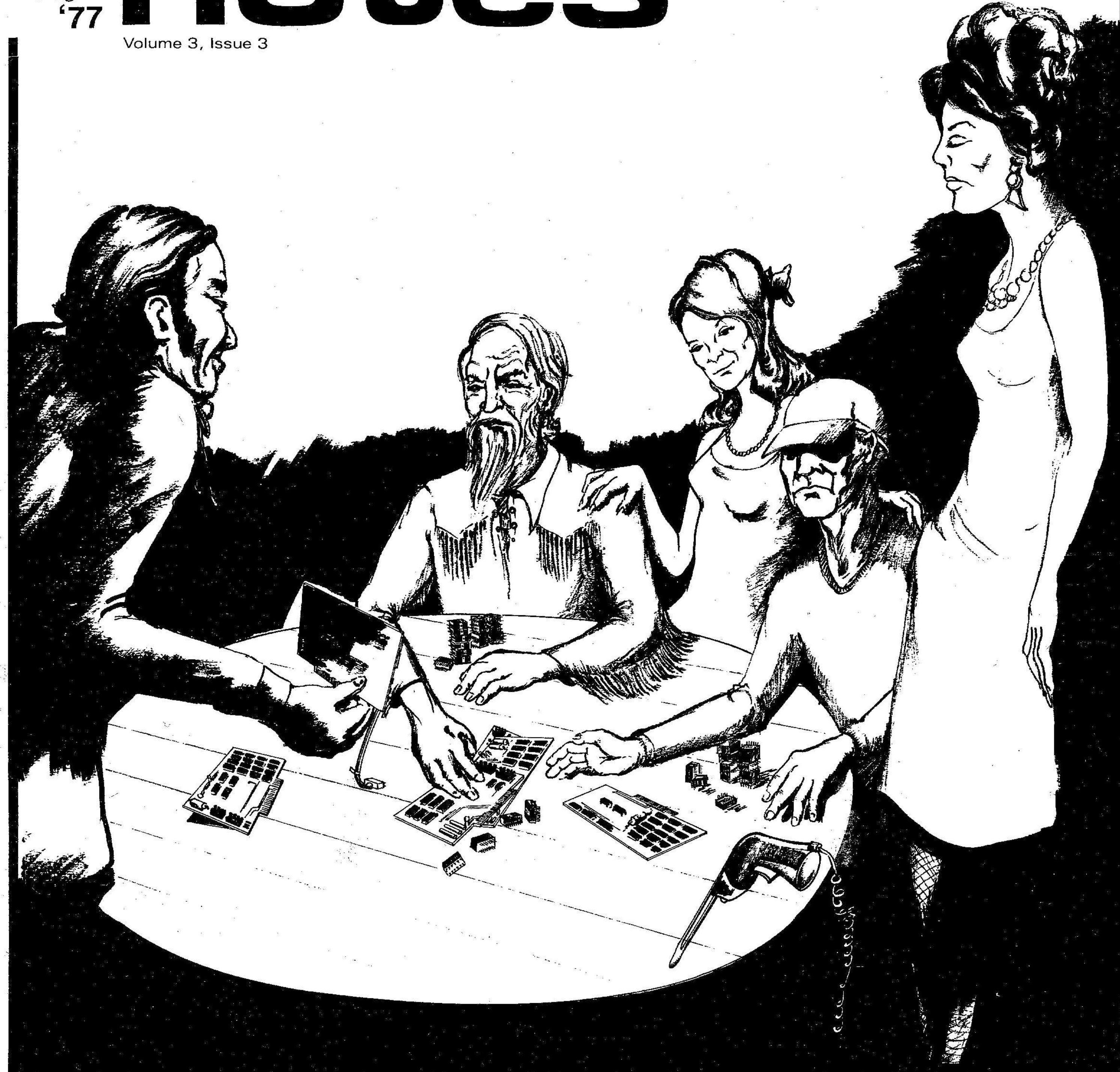
computer notes

50¢

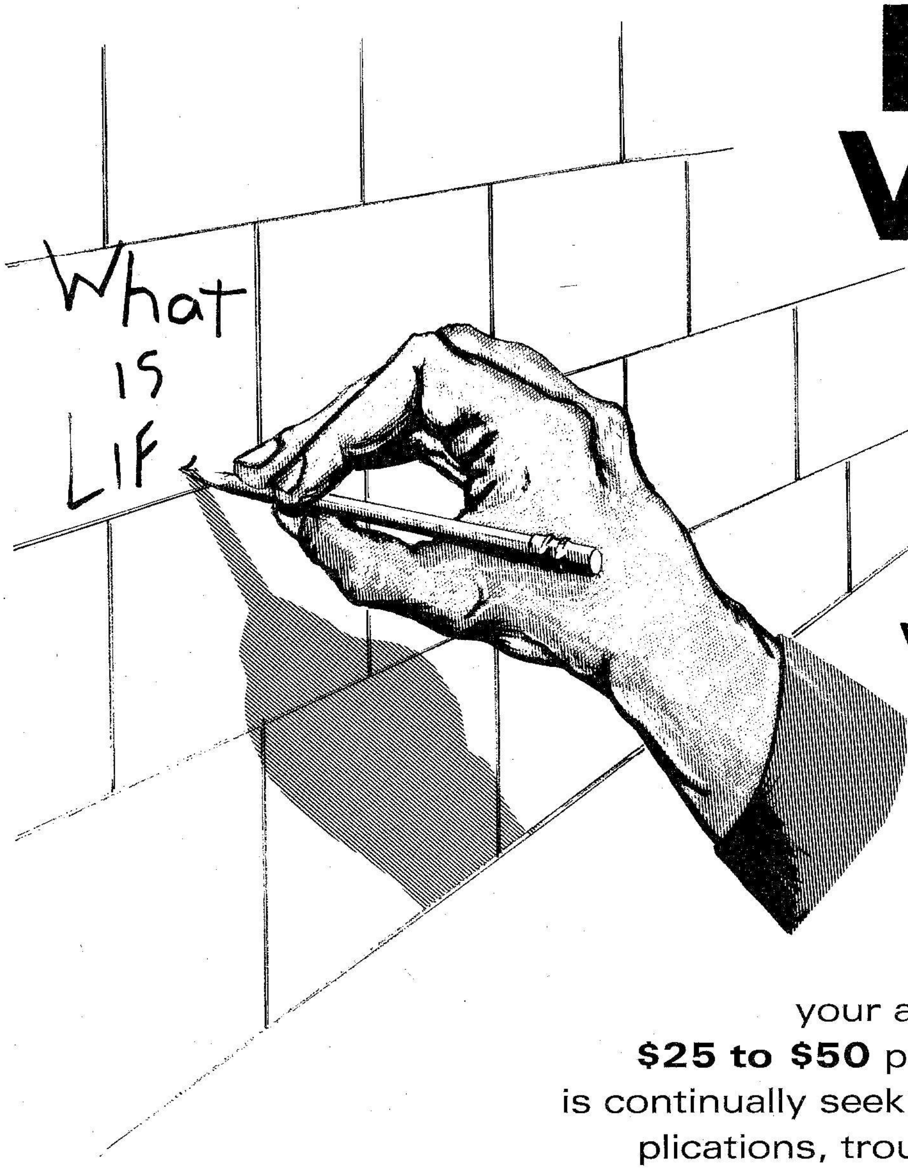
August '77

Volume 3, Issue 3



Special Insert: NEW ALTAIR™ HANDS-ON SAVINGS.

MAINFRAMES REDUCED UP TO 26%



DON'T WRITE IT ON THE WALL

Type it double-space on 8-1/2 x 11 paper and send it along to **Computer Notes.** If your article is accepted, you can get **$25 to $50** per typeset magazine page. **CN** is continually seeking quality manuscripts on applications, troubleshooting, interfacing, software and a variety of other computer-related topics.

All articles are subject to editing to fit space requirements and content needs of our readership. Payment for articles which are accepted will be sent upon publication.

Articles submitted to **C.N.** should be typed, double-space, with the author's name, address and the date in the upper left-hand corner of each numbered page. Authors should also include a one-sentence autobiographical statement about their job, professional title, previous electronic and/or computer expereince under the article's title. Authors should retain a copy of each article submitted.

All illustrations, diagrams, schematics and other graphic material should be submitted in black ink on smooth white paper. Prints and PMT's are acceptable.

COMPUTER NOTES is published monthly by MITS, Inc., 2450 Alamo SE, Albuquerque, NM, 87106, (505) 243-7821. A free year's subscription is included with every purchase of an Altair™ computer. Regular subscriptions can be ordered from the MITS Customer Service Dept. for $5 per year in the U.S. and $20 per year for overseas. Single copies are available for 50¢ each at all Altair Computer Centers. Send articles, questions, comments and suggestions to Editor, *COMPUTER NOTES*, MITS, Inc.

a subsidiary of **Pertec Computer Corporation**
2450 Alamo S.E., Albuquerque, New Mexico 87106

ALTAIR DISK OPERATING SYSTEM NOW AVAILABLE

By Gale W. Schonfeld
MITS

The Altair Disk Operating System (DOS) is a highly sophisticated assembly language development package which includes five major systems programs plus several miscellaneous programs to aid the user in software development.

The following programs are included in the system.

The Monitor

The DOS Monitor is the main control program for the package. When the system is powered up, the Monitor is loaded and remains resident in memory at all times during power-on. The Monitor is also used to load and execute the DOS system programs and user programs.

The system Monitor contains all necessary Input/Output (I/O) routines to support the majority of Altair computer products (2SIO, Disk, SIO, ACR, 4PIO, PIO). Programmers no longer need to insert I/O routines in their programs, because the Monitor may be called only for the necessary routines.

The Text Editor

The DOS TEXT Editor is used primarily to create and maintain assembly language program files. The Editor facilitates (1) insertion, (2) deletion and replacement of program lines, (3) in-line editing with the use of the ALTER command, such as adding, deleting or modifying characters within a line and (4) paging. The paging commands allow the user to minimize the usage of memory by sequentially loading one page of program test into memory at a time.

The Assembler

The DOS Assembler converts an assembly language program to machine language in two passes. In the first pass the Assembler reads the assembly code and assigns addresses to all the symbols. The second pass re-reads the program and converts the mnemonics and symbolic addresses to their machine language equivalents.

The DOS Assembler includes a set of pseudo-op instructions which reserve storage space, define contents of memory locations and control the parameters of the Assembler's operation.

The Linking Loader

The DOS Linking Loader is used to link the relocatable object code modules produced by the Assembler. It performs several functions, such as loading the modules and making sure that bytes of the module are not accidentally destroyed by an overlay of a subsequent module. (Intentional overlaying may be achieved by changing the default load address of the module.) It also makes connections between all external references and the addresses to which they refer, and prints a list of those that have undefined addresses. The Linking Loader can also search the disk for files to resolve undefined references.

The Debug

The DOS Debug package is designed primarily to control program execution during check-out. This system program provides several commands which allow the contents of memory locations, registers and flags to be displayed or modified. It also allows the user to insert, display and remove breakpoints to initiate pauses in program execution. Debug may also be used to begin program execution at any address or breakpoint.

The following miscellaneous programs are provided with the DOS software.

INIT

Allows the user to re-initialize the system (memory size, number of disks, number of disk files, etc.) without having to reload.

CNS

Allows the user to change his console device to another terminal.

Continued

EDITOR
Andrea Lewis

ASSISTANT EDITOR
Linda Blocki

PRODUCTION
Al McCahon
Steve Wedeen
Beverly Gallegos
Alice Regan

CONTRIBUTORS
Dave Antreasian
Mark L. Chamberlin
Pat Diettmann
Susan B. Dixon
Thomas Durston
Bruce Fowler
Bennett Inkeles
Ken Knecht
Richard F. Ranger
Gale W. Schonfeld
Glenn Wolf

a subsidiary of
Pertec Computer Corporation
2450 Alamo S.E.
Albuquerque, New Mexico 87106

IN THIS ISSUE

ALTAIR DISK OPERATING SYSTEM NOW AVAILABLE

Continued

SYSENT
A system program file that contains the addresses of several Monitor routines that are available for user programs.

LIST
A BASIC language routine to be used with Altair Disk Extended BASIC; allows DOS Assembler listing files to be printed on a line printer. Altair DOS also includes (1) a copy utility program, which enables files to be copied from one disk to another, (2) 27 error codes in the Monitor and 16 in the Assembler and (3) the ability to use Absolute, Relative, Common, Data or External addressing modes with the Assembler. Although the object code produced by the Assembler is listed in octal, constant addresses used in the Assembler source program may be expressed in octal, decimal or hexadecimal.

The Altair Disk Operating System software can be loaded with the Disk Bootstrap Loader PROM (DBLP) currently being used by Altair Disk BASIC users. However, to use it with DOS requires the hardware configuration to have at least 20K of RAM memory (since the DBLP loads into RAM just under that point). A new version of the DBLP which will allow loading in 16K will be released soon so that the user need not have a 20K system.

DETAIL

(1) Hardware Required:
- -Altair 8800 series mainframe and CPU
- -Minimum I6K of Altair RAM
- -Altair 88-DCDD
- -Altair I/O

(2) Price:
- -$200 with minimum system comprised of Altair equipment.
- -$500 without minimum system

(3) Delivery:
- -60 days after receipt of order

(4) Format:
- -DOS is available only on diskette and includes a paper tape and a cassette tape bootloader plus the documentation

Altair Timesharing BASIC Ideal for Educational and Scientific Applications

By Susan B. Dixon
MITS

Altair Timesharing BASIC, carefully developed to allow as many as eight users independent and simultaneous access to the facilities of Altair BASIC, possesses many of the high-level system design features usually associated with larger computers. The following description of some of these features illustrates why Altair timesharing BASIC is a cost-effective system ideally suited for educational and scientific applications.

Security Provisions

Timesharing BASIC is available in two versions—Altair Timesharing BASIC and Altair Timesharing Disk BASIC. Both are a superset of Altair Extended BASIC and are specially modified to provide each user absolute security for the job being executed. As a security provision, the following features of Altair Extended BASIC—USR, WAIT, IMP, OUT, PEEK and POKE—are disallowed in both versions of Timesharing BASIC.

There are two means of protecting disk programs through the use of passwords in Altair Timesharing Disk BASIC. A four character password must be used for READing a program. The program cannot be KILLed or NAMEd without a password. Passwords allow the user to read and modify a program. They are also required to access the functions MOUNT and UNLOAD. A password and argument are needed to access . DISKINI. This special password system protects the file system and prevents users from altering the programs of other users.

Saving and Loading Programs

In Timesharing BASIC programs may be loaded and stored on paper tape only. But program files may be SAVEd and LOADed in Timesharing Disk BASIC. However, there are currently no capabilities for creating sequential or random data files.

A fixed memory size and the I/O address of each terminal is established by the operator during the initialization dialogue. Both versions of Timesharing BASIC have the flexibility to allow the operator to establish different memory sizes (above a minimum of IK for each job).

Terminal Transfer of Jobs

Procedures for CONSOLEing are easier than those used in the regular versions of Altair BASIC. The user can transfer control of a job from the present terminal to a specified terminal by entering CONSOLE (address).

Reloading Timesharing BASIC

Due to the complex nature of the interrupt operators, EXAMINE 0 will not restart Timesharing BASIC. BASIC must be reloaded if the STOP switch on the front panel is activated.

Interface Capabilities

Terminal interface is governed by the 2SIO board. Any terminal which can be connected to the 2SIO board (RS232, TTY with 60/25 ma current loop, TTL) can be utilized for both versions of Altair Timesharing BASIC.

Output

Both versions of Timesharing BASIC are supported only by the Altair 88-2SIO, Q70 and C700 line printer interface boards. When a program is generating output, characters are stored in an output buffer. If the user is "swapped out"* and the I/O device is operating at a high baud rate, the buffer may empty more rapidly than it is filled. Printing will then cease temporarily until the user is "swapped" back in.

NOTE: The term "swapped out" usually refers to a user being swapped to a disk. Although technically this does not occur in Altair Timesharing BASIC, it is not within the scope of this article to describe the actual process.

MINIMUM HARDWARE REQUIREMENTS
- -Altair 8800 series mainframe and CPU
- -A minimum 32K RAM
- -88-VI/RTC (Vector Interrupt/Real Time Clock)
- -Maximum of four 2SIO boards to interface with terminals
- -88-DCDD (Disk Controller Disk Drive) Altair T/S Disk BASIC
- -88-PMC (PROM Memory Card) Altair T/S Disk BASIC

The above hardware is essential for the

MITS software specialist Chuck Vertrees explains the special modification included in Altair Timesharing BASIC and Altair Timesharing Disk BASIC to an interested user at the recent NCC in Dallas.

operation of Timesharing BASIC. The 88-PMC (PROM Memory Card) is necessary if the Multibootloader or Disk Bootloader PROMs are utilized for easier loading.

EDUCATIONAL APPLICATIONS

Altair Timesharing BASIC's rigid security features and rapid keyboard response coupled with the many powerful capabilities of Altair Timesharing BASIC ideal for classroom use.

Educational institutions currently sharing costly time on a larger system or utilizing smaller systems for demonstration will find that Altair Timesharing BASIC can provide an opportunity for actual on-line programming experience. As many as eight students can write, debug and run programs simultaneously with no discernable keyboard delay.

Altair Timesharing BASIC can be a valuable tool for computer-aided instruction. Simple programs such as the one below can provide younger students with a unique and stimulating approach to dull, repetitous drills.

```
10 B=0
20 INPUT "WHAT IS 3*4;A$
30 IF A$="I2" THEN 80
40 PRINT "WRONG!!"+CHR$(7)
50 B=B+1
60 PRINT "TRY AGAIN"
70 GOTO 20
PRINT "RIGHT YOU MADE"; B; "MISTAKE(S)"

WHAT IS 3*4?  I3
WRONG!!
TRY AGAIN.
WHAT IS 3*4?  12
RIGHT. YOU MADE 1 MISTAKE(S)
```

This program can be expanded to include the entire set of multiplication tables. The program informs students as to the number of mistakes they make. This type of drill can be set up in a modular form with different modules for various subjects, such as history, spelling, biology, etc.

Computer-aided instruction goes far beyond these simple drills to the teaching of heuristic logic essential to the scientific method.

AVAILABILITY AND COST

Altair Timesharing BASIC is already available on audio cassette, and Altair Timesharing Disk BASIC will be available by the middle of this month. Both versions of BASIC must be supported by the hardware listed in this article.

The suggested retail price of Altair Timesharing BASIC is $600. For users currently operating under another version of Altair BASIC, an upgrade charge (the difference in price between your current version of BASIC and Timesharing BASIC) plus a $25 copy fee will be made.

Altair Timesharing Disk BASIC sells for a suggested retail price of $750. An upgrade cost plus a $35 copy fee will be charged to users currently operating with any other versions of Altair BASIC.

MITS Production Department Emphasizes Quality Construction Not Mass Production

Production Manager Fred Sanchez watches technician Julia Sena perform delicate touch-up procedures on an Altair 2SIO board.

By Susan B. Dixon

In this age of mass-produced merchandise, quality craftsmanship is often difficult to find and is usually associated with non-mechanical objects. MITS Altair microcomputers, carefully constructed by hiqhly skilled production personnel using advanced automated equipment, are one of the exceptions.

Rigid inspections, meticulous quality control and the coordinated efforts of 60 production workers result in the finely crafted Altair microcomputer series. "The Production Department assembles an average of I0 Altair 8800b microcomputers everyday," said Production Manager Fred Sanchez. To achieve this efficient production flow, Sanchez said the department is divided into four areas: assembly, subassembly, quality control and testing.

The first step, the assembly process, begins with the insertion of all integrated circuit components, which are machine pressed onto the circuit board, he explained. In another automated process

ALTAIR 8800 BOARDS CREATE TROUBLESHOOTING BREAKTHROUGH

By Bennett Inkeles

MITS recently developed two new products with plug-in options which significantly simplify troubleshooting of Altair 8800 systems. When the Diagnostic Card (88-DC) and Switchable Extender (88-SE) are used in relation to other waveforms SE) are used in conjunction with the oscilloscope, suspected problem signals may be viewed in relation to other waveforms for correct timing relationships. Dip switch packages for signal line selection aid in isolating system faults.

The Diagnostic Card (88-DC) is capable of displaying eight signals on any inexpensive oscilloscope. Any combination of data, address and control lines can be selected (except where redundancy of dip switches causes the computer to become inoperative).

Other features include an Eight-Bit-to-Octal Decoder/Display for program inspection. Users may examine the program on either data input or output lines. For bus and card regulator voltages, a digital voltmeter (+1 per cent accuracy) is supplied, which is switch selectable between +8, +18, -18 bus voltages and External (+9.99 full scale range). An output synchronization jack is provided for display referencing of data and control signals. Two BNC cables (not included) are required for oscilloscope connections. A test lead is included for external voltage measurements.

Additional capabilities include External input through a 10-pin right angle connector. Troubleshooting of individual boards is achieved by entering signals to the Diagnostic Card by means of an IC test clip. This configuration is particularly useful in monitoring critical timing relationships.

The Switchable Extender (88-SE) assists in localizing problems by switching suspected problem lines out of the circuit. By using dip switches, areas of question may be easily pinpointed.

There are 10 dip switch packages. Each contains eight switches which represent all of the control, data and address lines. Six of the switches are uncommitted and can be implemented as the user desires. Vector interrupt capabilities are available on the card and are activated by installing the required jumpers.

Both the 88-DC and the 88-SE are available only assembled. Suggested retail prices for the 88-DC and the 88-SE are $365 and $125 respectively.

small electronic components such as resistors and capacitors are inserted with an electrical component lead former. "This inserts the component and bends the leads," he said. Gesturing to a large vat of bubbling solder, Sanchez said, "Hand soldering of all components is no longer part of the production process. Instead, this wave solder machine saves four to five hours of labor previously needed to solder all the components on a single board." A worker places the board in an adjustable holder, and a conveyor then guides the board through a flow of bubbling solder.

Sanchez said that some boards, such as the mother board for the Altair 8800, must be specially modified to withstand the automated soldering process.

"The boards are inspected after they leave the wave solder machine," he continued. "Any boards which require touch-up of things like solder bridges and cold-solder joints are done here," he said, pointing to a long table with tangled bright lights where 15 women were scrutinizing boards and performing delicate solder touch-up.

Sanchez said production also assembles some "custom units," which require extensive hand soldering. "Production makes many of the prototypes of MITS products. We also make certain special modifications for customers," he added. "For example, some units from overseas customers need to be specially adapted to run under a different current. Hand soldering is also used for all heat sensitive components such as LED's and switches," he noted.

The second aspect of production is the subassembly of various parts of the main frame. "Here the women put together the back and front panels, the cross-members and motherboard," Sanchez said. The subassembly of the main frame is combined with the assembly of the various mechanical components, such as the power supply and fan, he explained.

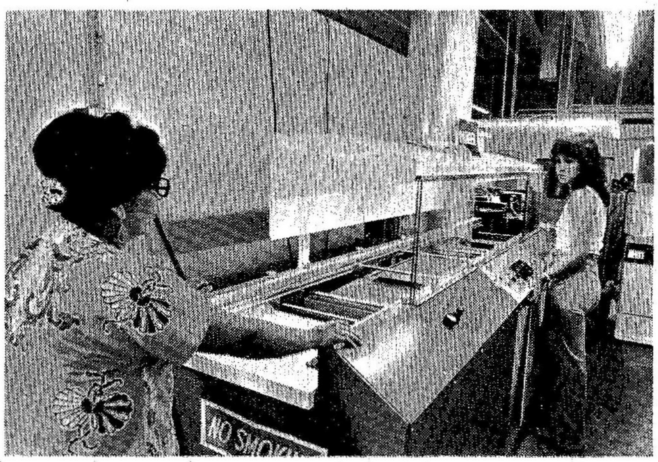
The third step of the production process is quality control. "Before the boards are installed, they are returned to Quality Control for a final visual inspection," he said, as he walked over to a table where six women were intently inspecting boards under large light-ringed lenses.

Sanchez said the next step in production is the addition of all cables, a thorough inspection of all subassemblies and then a preliminary test. "If there are no problems with the unit, it goes back to the mechanical assembly section to be completely enclosed in a case," he said.

Each completed unit makes a final trip to Testing, the fourth area of production, where it is plugged in and left running continuously for 48 hours. "This is called a 'burn-in test'. It insures that users receive only units which operate trouble free, and contain no defective components," Sanchez explained.

Production is closely coordinated with MITS marketing department. Sanchez said a monthly 'forecast' issued by Marketing projects the number of each product which will be sold in that month. Production then gears their efforts to produce an inventory which will fill the sales forecast. "Right now we're running with no backlog of orders, and that's really a credit to production," Sanchez said with a smile. "But that's because people involved in the assembly process (95% women) are all first rate and have had previous electronic assembly experience," he added.

In the final step of the assembly process, the finished Altair products which have completed the numerous tests and inspections are then carefully double boxed. As a safety precaution, even those products stored in inventory at MITS are doubled boxed, Sanchez noted.



Wave solder technicians Genieve Lucero (left) and Tonie Sanchez guide Altair PC boards through the wave soldering machine.

Electronics assembly technicians Vickie Howell (foreground) and Luz Alba construct the mainframe of an Altair 8800b.

SOFTWARE

Program Control at Your Fingertips

By Mark L. Chamberlin

The Problem

Have you ever watched output from a program spew forth at 9600 baud, rapidly scrolling off the top of your CRT screen? For those of us who cannot read and fully comprehend at 11,500 words per minute, this represents somewhat of a problem. (Those who can have already finished this article.) Or perhaps you've written a program which looped for longer than it should have. Maybe it was hung in an infinite loop. (But who has the patience to determine this empirically?)

Such problems plague even experts. Although the situations may vary, they stem from a single problem—the executing program is out of control. The obvious hardware solution to this problem is to reach for the HALT switch on the front panel. This solution is not only inconvenient and rather inelegant, but it doesn't provide the flexibility required in many situations.

This article describes a software design which allows users to exercise various controls over an executing program without ever leaving the terminal. It deals primarily with assembly code programs, but some of the information should also prove interesting to BASIC programmers. Although the particular implementation discussed is for the Altair 680b, the design is general enough to be implemented on most microcomputers and forms a subsystem which can be easily incorporated into existing software systems or designed into new systems.

Towards a Solution

Programmers generally keep their fingers in close proximity to the terminal keyboard, ready to fire off the next command when the prompt appears. Therefore, the keyboard is the most convenient place from which to control a program. One way to provide program control from the keyboard is to assign control functions to special keyboard codes. When one of the special codes is typed, its associated control function is performed. There's nothing new about this idea. Anyone who has ever used DEC computers knows that typing "control-C" causes control to be returned to the system monitor. Altair BASIC contains this same feature—typing "control-C" causes an executing BASIC program to return to command mode.

A list of the special keyboard codes and their associated control functions to be supported here is given below. The list is a subset of those supported by the DEC System-10 and is a complete list of those supported by Altair 8800 Extended BASIC.

1. Control-C

Typing "control-C" causes control to be returned to the system monitor. It is typically used when a program gets "lost" or when the programmer decides to terminate a program prior to completion.

2. Control-S

Typing "control-S" causes program execution to be suspended. Program output may then be read from the CRT prior to allowing program execution to proceed.

3. Control-Q

Typing "control-Q" causes execution of a suspended program to be resumed. It is used to continue program execution after a "control-S" has been typed.

4. Control-O

Typing "control-O" causes program execution to continue while inhibiting program output. Program output is inhibited until another "control-O" is typed or the program overides this control. This function is used when a program's terminal output is not required for a particular run. This is especially useful when a Teletype™ or other slow terminal is being used, as it speeds up overall throughput.

In order to gain some insight as to how these control functions can be implemented, let's examine the way Altair BASIC handles them. Prior to executing each line of a BASIC program, the Altair BASIC interpreter checks the terminal input status bit. If the status bit indicates that a character has been typed, BASIC reads the character and checks to see if it is a special control character. If it is, the appropriate action is taken. If it is not a special control character, the interpreter proceeds with the execution of the next BASIC program statement. A flowchart detailing this procedure is shown in Fig. 1.

Flowchart for
BASIC Control Character Handling

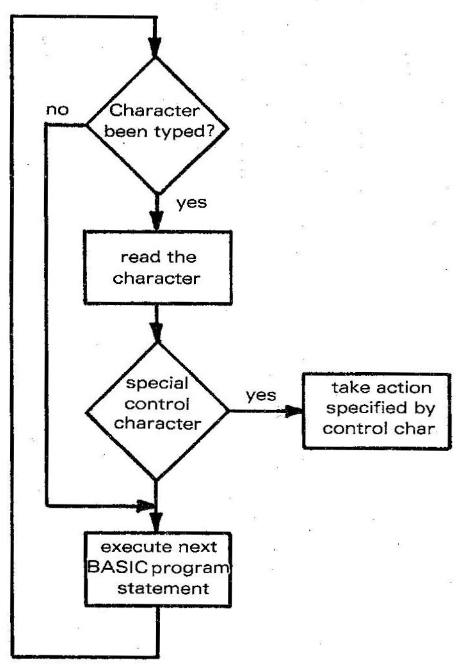


Figure 1

Although this method of polling the terminal on a regular basis proves adequate for the Altair BASIC interpreter, it is clumsy at best in the general case. For example, who would want to put keyboard checks into every loop in an assembly code program? How about forcing the computer to look for special control characters every time a key is typed, regardless of what the program is doing? This can be accomplished through

the use of program interrupts. Although the subject of program interrupts is a complex one, the basic concepts required here are fairly simple. Interrupts are used to force the computer to drop what it is doing and execute a special section of code referred to as an "interrupt service routine." When told to do so by the interrupt service routine, the computer proceeds with what it was doing before it was so rudely interrupted.

Conveniently enough, most computers have the capability to cause an interrupt when a key is typed on the terminal. The use of this keyboard interrupt capability forms the basis of the design set forth below.

The Design

The design consists of a set of assembly code subroutines. The primary routine is an interrupt service routine which checks for control characters every time a key is typed and takes the appropriate action when it sees control characters. In addition to the interrupt service routine, subroutines to input a character from the terminal, output a character to the terminal and initialize the system variables are included. Detailed descriptions and flow charts of each of the routines are given below.

1. INTSRV - The interrupt service routine (See Figure 2.)

Flowchart for INTSRV

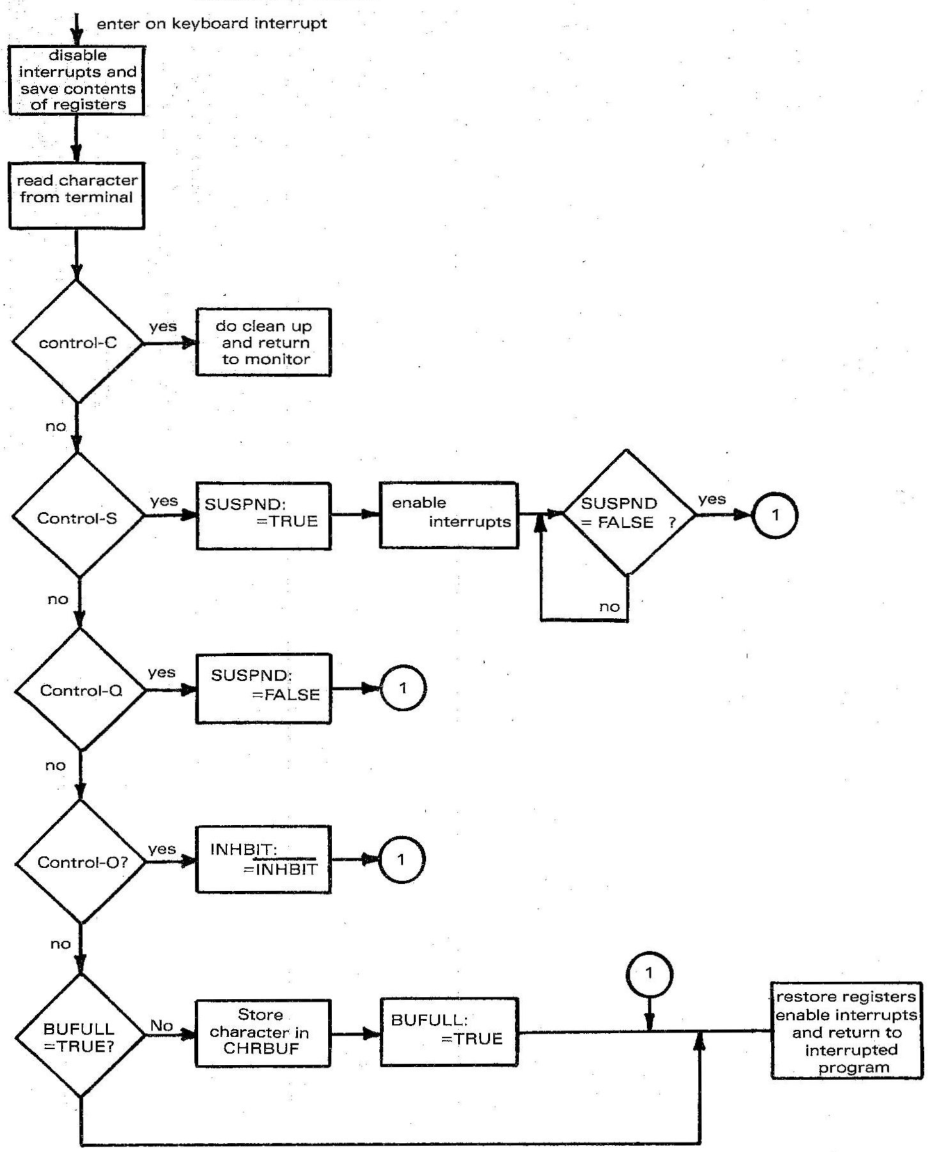


Figure 2

INTSRV is invoked (executed) each time a key is typed on the terminal. INTSRV reads the character from the terminal and checks to see if it is a special control character (control-C, control-S, control-Q, control-O). If it is, the appropriate action is taken as follows:

a. Control-C

If the character is a "control-C", then control is returned to the system monitor. It may be desirable to do some housecleaning, such as resetting the system variables, prior to returning to the monitor.

b. Control-S

If the character is a "control-S", then the following action is taken:

— The program suspension flag, SUSPND, is set to TRUE.

— Interrupts are enabled, thereby allowing another keyboard interrupt to invoke INTSRV

—The processor loops until SUSPND goes FALSE. In order for SUSPND to go FALSE, a "control-Q" must be typed. This is explained in greater detail below.

— When SUSPND goes FALSE, control is returned to the suspended program, which resumes where it left off. Notice that a "control-C" will take effect even while a program is suspended.

c. Control-Q

If the character is a "control-Q", the program suspension flag, SUSPND, is set to FALSE. Control is then returned to the interrupted program. This usually means that control is returned to the loop that is waiting for SUSPND to go FALSE. The net effect is that the suspended program is resumed. However, it is conceivable that someone might type a "control-Q" even though a "control-S" had not been typed. In this case, the "control-Q" has no real effect.

d. Control-O

If the character is a "control-O", the inhibit output flag, INHBIT, is complemented (set to TRUE if it was previously FALSE; set to FALSE if it was previously TRUE). Therefore, typing "control-O" inhibits or enables program output, depending on the previous state of the inhibit output flag. INHBIT is usually set to FALSE during initialization. Programs may set INHBIT to FALSE to insure that important messages such as prompts are seen by the user.

If the character read from the terminal is not one of the special control characters, then INTSRV checks the input buffer full flag, BUFULL. If BUFULL is TRUE it

indicates that the single character input buffer, CHRBUF, is full; INTSRV will then discard the character it just read from the terminal. This prevents the overwriting of the last character typed before the program has had a chance to use it. If BUFULL is FALSE, then INTSRV stores the character it read from the terminal in CHRBUF and sets BUFULL to TRUE. (The functions of BUFULL and CHRBUF are further explained in the description of the input character routine.) After dealing with the character, INTSRV returns control to the interrupted program.

2. INCHAR - The input character routine (See Figure 3.)

Flowchart for INCHAR

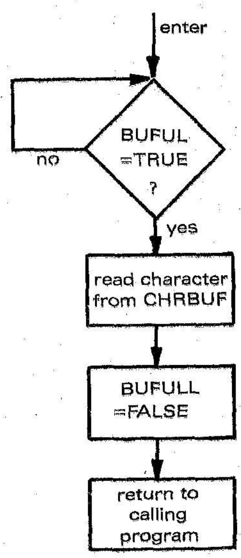


Figure 3

Most programs call a subroutine to get a character from the terminal. Usually such routines loop until the terminal input status bit indicates that a character has been typed. Then they read the character from the interface and pass it back to the calling program. Reading the character from the terminal interface automatically resets the input status bit to indicate that the character in the interface is not current.

Since INSRV reads every character before any other routine does, the terminal input status bit always indicates that the character in the interface is not current. This is why INTSRV stores the characters it reads in CHRBUF and sets the flag BUFULL to TRUE. As a result, instead of waiting on the terminal input status bit and reading a character from the terminal interface,

```
                          NAM        CONTRL
                          OPT        P,M
                    *
                    * CONTROL CHARACTER HANDLER ROUTINES
                    *
                    * MARK L CHAMBERLIN  6/30/77
                    *
                    * SOME SYMBOL DEFINITIONS
                    *
          0003      CONTC  EQU       'C-@100   CONTROL-C CODE
          0013      CONTS  EQU       'S-@100   CONTROL-S CODE
          0011      CONTQ  EQU       'Q-@100   CONTROL-Q CODE
          000F      CONTO  EQU       'O-@100   CONTROL-O CODE
          F000      TRMSTA EQU       $F000     ACIA CONTROL/STATUS
          F001      TRMDAT EQU       TRMSTA+1  ACIA DATA
          FFD8      MONIT  EQU       $FFD8     MONITOR'S ENTRY POINT
          00F3      ECOFLG EQU       $00F3     MONITOR'S ECHO FLAG
                    *
                    * ROUTINES START AT 16K
                    *
     4000                 ORG        $4000
                    *
                    * FLAGS ARE TRUE WHEN NONZERO AND FALSE WHEN ZERO
                    *
     4000 0001      SUSPND RMB       1         PROGRAM SUSPENSION FLAG
     4001 0001      INHIBT RMB       1         INHIBIT OUTPUT FLAG
     4002 0001      BUFULL RMB       1         BUFFER FULL FLAG
     4003 0001      CHRBUF RMB       1         INPUT CHARACTER BUFFER
                    *
                    *
                    * INTSRV - THE INTERRUPT SERVICE ROUTINE
                    * INVOKED EACH TIME A CHARACTER IS TYPED ON KEYBOARD
                    *
     4004 B6 F001   INTSRV LDA A     TRMDAT    READ THE CHARACTER
     4007 84 7F            AND A     #$7F      RESET THE PARITY BIT
     4009 CE 4000          LDX       #SUSPND   SET UP TO USE INDEXED MODE
     400C 81 03            CMP A     #CONTC    IS CHAR A CONTROL-C?
     400E 26 05            BNE       NOTC      NO
     4010 6F 01            CLR       1,X       YES,SET INHBIT TO FALSE
     4012 7E FFD8          JMP       MONIT     AND GO TO PROM MONITOR
                    *
     4015 81 13     NOTC   CMP A     #CONTS    IS CHAR A CONTROL-S?
     4017 26 09            BNE       NOTS      NO
     4019 A7 00            STA A     X         YES, SET SUSPND TO TRUE
     401B 01               NOP                 ENABLE INTERRUPTS
     401C 0E               CLI
     401D 6D 00     HANG   TST       X         HAS SUSPND GONE FALSE YET?
     401F 26 FC            BNE       HANG      NO, HANG AROUND
     4021 3B               RTI                 YES, RETURN FROM INTERRUPT
                    *
     4022 81 11     NOTS   CMP A     #CONTQ    IS CHAR A CONTROL-Q?
     4024 26 03            BNE       NOTQ      NO
     4026 6F 00            CLR       X         YES, SET SUSPND TO FALSE
     4028 3B               RTI                 AND RETURN FROM INTERRUPT
                    *
     4029 81 0F     NOTQ   CMP A     #CONTO    IS CHAR A CONTROL-O?
     402B 26 03            BNE       NOTO      NO
     402D 63 01            COM       1,X       YES, COMPLEMENT INHBIT
     402F 3B               RTI                 AND RETURN FROM INTERRUPT
                    *
     4030 6D 02     NOTO   TST       2,X       CHAR ISN'T CONTROL CHAR
     4032 26 04            BNE       DISCRD    BUFFER FULL - DISCARD CHAR
     4034 A7 03            STA A     3,X       PUT CHAR IN CHRBUF
     4036 63 02            COM       2,X       AND SET BUFULL TO TRUE
     4038 3B        DISCRD RTI                 RETURN FROM INTERRUPT
                    *
                    *
                    * INCHAR - THE INPUT CHARACTER ROUTINE
                    * RETURNS CHARACTER READ IN B
                    * ALL OTHER REGS PRESERVED
                    *
     4039 7D 4002   INCHAR TST       BUFULL    CHAR IN CHRBUF?
     403C 27 FB            BEQ       INCHAR    NO, WAIT FOR ONE
     403E F6 4003          LDA B     CHRBUF    READ THE CHAR
     4041 7F 4002          CLR       BUFULL    SET BUFULL TO FALSE
     4044 7D 00F3          TST       ECOFLG    SHOULD WE ECHO?
     4047 2A 01            BPL       OUTCHR    YES
     4049 39               RTS                 AND RETURN TO CALLER
                    *
                    *
                    * OUTCHR - THE OUTPUT CHARACTER ROUTINE
                    * IF INHBIT IS FALSE THEN INCHAR
                    * OUTPUTS THE CHAR PASSED TO IT IN B
                    * OTHERWISE CHARACTER IS NOT OUTPUT
                    * ALL REGISTERS PRESERVED
                    *
     404A 7D 4001   OUTCHR TST       INHBIT    INHIBIT OUTPUT?
     404D 26 0C            BNE       OUTRTS    YES, JUST RETURN
     404F 37               PSH B               SAVE THE CHAR
     4050 F6 F000   WAIT   LDA B     TRMSTA    WAIT FOR READY
```

New Hands-On Savings

**Share our lower production costs.
Mainframes now reduced up to 26%.**

Altair™

Announcing New Altair™ Hands-On Savings

Effective immediately: 14%-to-26% price reductions on the nation's first choice in μC systems.

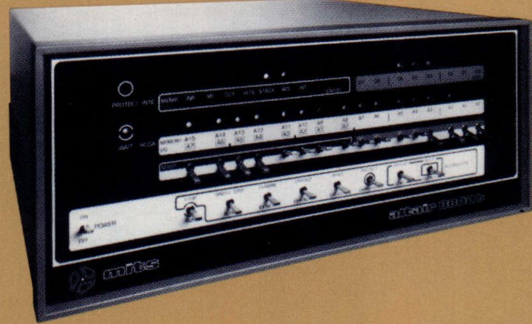
The cost of going first class has just come down. Way down. Now you can build the country's most popular personal computers into any system for significantly less: Altair mainframe prices are now dramatically reduced-*from $67 to $255 off!*

Big hands-on savings for your own hands-on operation. Without giving an inch on our strict quality and performance standards—the same standards that have made Altair microcomputers the recognized leader, in homes, hobbies and small businesses.

But in these days of skyrocketing costs, how can we do it?

By new corporate design. Our recent acquisition by Pertec Computer Corporation has produced precisely what we were looking for: major manufacturing efficiencies, improved material handling and inventory procedures, and more effective financial management and cost controls.

That's how we're producing the same Altair systems the market counts on us for, but at a much lower cost to us. And we want to pass the same kind of savings on to our customers. Now that PCC is sharing its solid corporate strengths with us, it's our turn to share the advantages with you. In meaningful savings you can get your hands on...today.



Hands-On Savings for 8800b Mainframe: Reductions up to $180

The pacesetter in personal microcomputer systems, the Altair 8800b is a superb general purpose, byte-oriented machine, supporting up to 65K of directly addressable memory. Multi-interface options offer maximum system flexibility and expansion capabilities.

Includes: 8080-A CPU board, Interface buffer board, Case, Power Supply, Fan, Full front panel, 18-slot motherboard (EC18), Manual. (Memory not included.) Plus a one year membership in the ALTAIR User Group with return of warranty card to factory.

	Was	Now	Hands-On Savings
ALTAIR 8800b KIT with EC18: 2 edge connectors, 4 card guides.	$ 875	$ 750	$125 (14%)
ALTAIR 8800b ASSEMBLED with EC18: 12 slots filled (12 edge conn., 24 card guides).	$1175	$ 995	$180 (15%)
18 slots filled (18 edge conn., 36 card guides).	$1250	$1070	$180 (14%)

Hands-On Savings for 8800b-Turnkey: Reductions up to $255

Easy to use, this advanced Turnkey microcomputer is geared to go. Ideal for software buffs and business applications not requiring front panel control.

Includes: CPU board, Case, Power Supply, Fan, Turnkey Front Panel, 18-slot motherboard (EC18). Turnkey Module Board contains a serial input/output

(SIO) interface, 1K of RAM, provision for 1K of PROM and sense switches—all expandable. With a one year's membership in the ALTAIR User Group with return of warranty card to factory.

	Was	Now	Hands-On Savings
ALTAIR 8800b-TURNKEY KIT with EC18: 1 edge connector, 2 card guides	$ 995	$ 795	$200 (20%)
ALTAIR 8800b-TURNKEY ASSEMBLED with EC18: 12 slots (12 edge conn., 24 card guides)	$1250	$ 995	$255 (20%)
18 slots (18 edge conn., 36 card guides)	$1325	$1070	$255 (19%)

Hands-On Savings for 680b Mainframe: Reductions up to $130

New pricing makes this compact, low-cost microcomputer still more economical for home computing and process control applications. Complete computer capabilities on a single space-saving mainboard: PROM Monitor for immediate program load/run in both BASIC and Assembly languages; ACIA for software controlled interfacing to serial terminal devices.

Includes: Case, CPU board with 1K static RAM, 256 byte PROM Monitor, sockets for an additional 768 bytes of 1702A PROMs (extra PROMs not included), one ACIA serial interface port. Complete front panel and power supply. (Expander card NOT included.) One year ALTAIR User Group membership with return of warranty card to factory.

	Was	Now	Hand-On Savings
ALTAIR 680b KIT: 680b Computer WITHOUT expander card	$ 446	$ 395	$ 67 (15%)
ALTAIR 680b ASSEMBLED: 680b Computer WITHOUT expander card	$ 625	$ 495	$130 (20%)

Hands-On Savings for 680b-Turnkey: Reductions up to $160

The Altair Turnkey version of the small 680b: in big demand for software dedicated systems, where full front panel operations are not required. When used with a terminal, the system's PROM Monitor makes front panel control unnecessary.

Includes: all specifications of the 680b Mainframe, except complete front panel is NOT included. Front Panel provided for the 680b-Turnkey includes: Power indicator, Reset switch, and Run/Halt switch. One year ALTAIR User Group membership with return of warranty card to factory.

	Was	Now	Hands-On Savings
ALTAIR 680b-TURNKEY KIT: 680b-T Computer Without expander card	$425	$350	$ 75 (17%)
ALTAIR 680b-TURNKEY ASSEMBLED: 680b-T Computer WITHOUT expander card	$610	$450	$160 (26%)

All new and former prices listed are based on the Altair Suggested Retail Price Lists: individual prices may vary.

Altair User Group membership provides a free subscription to *COMPUTER NOTES* and access to the User Group Software Library.

Get Hands-On flexibility in Altair™ peripherals, software and plug-in options.

Available from PCC: a versatile range of Altair add-ons and software. Disk storage, CRT terminals, printers. Business accounting, word processing and inventory management systems. Expansion memory and interface boards. Full lists and details now at your Altair dealer, and in the Altair line brochure.

Altair Computer Centers

For expert assistance in planning and implementing your microcomputer system, visit the qualified people at one of these Altair Computer Center locations.

TUCSON, AZ 85711
4941 East 29th Street
(602) 748-7363

BERKELEY, CA 94710
1044 University Avenue
(415) 845-5300

SANTA MONICA, CA 90401
820 Broadway
(213) 451-0713

DENVER, CO 80211
2839 W. 44th Avenue
(303) 458-5444

ATLANTA, GA 30305
3330 Piedmont Road, N.E.
(404) 231-1691

PARK RIDGE, IL 60068
517 Talcott Road
(312) 823-2388

ANN ARBOR, MI 48104
310 East Washington Street
(313) 995-7616

MADISON HEIGHTS, MI 48071
505-507 West 11 Mile Street
(313) 545-2225

EAGAN, MN 55122
3938 Beau D'Rue Drive
(612) 452-2567

ST. LOUIS, MO 63130
8123-25 Page Boulevard
(314) 427-6116

LINCOLN, NE 68503
611 N. 27th Street, Suite 9
(402) 474-2800

CHARLOTTE, NC 28205
1808 East Independence Boulevard
(704) 334-0242

ALBUQUERQUE, NM 87110
3120 San Mateo, North East
(505) 883-8282, 883-8283

ALBANY, NY 12211
269 Osborne Road
(518) 459-6140

NEW YORK, NY 10018
55 West 39th Street
(212) 221-1404

DAYTON, OH 45414
5252 North Dixie Drive
(513) 274-1149

TULSA, OK 74135
110 The Annex
5345 East Forty First Street
(918) 664-4564

BEAVERTON, OR 97005
8105 South West Nimbus Avenue
(503) 644-2314

DALLAS, TX 75234
3208 Beltline Road, Suite 206
(214) 241-4088 Metro—263-7638

HOUSTON, TX 77036
7302 Horwin Drive, Suite 206
(713) 780-8981

RICHMOND, VA 23230
1905 Westmoreland Street
(804) 355-5773

SPRINGFIELD, VA 22150
6605 A Backlick Road
(703) 569-1110

CHARLESTON, WV 25301
Municipal Parking Building
Suite 5
(304) 345-1360

ALTAIR™ Microcomputers
Pertec Computer Corporation
Microsystems Division

2450 Alamo S.E.
Albuquerque, New Mexico 87106
Telephone (505) 243-7821

102922

```
4053 C4 02          AND B     #2
4055 27 F9          BEQ       WAIT
4057 33             PUL B               RETRIEVE CHAR
4058 F7 F001        STA B     TRMDAT    SHIP IT OUT
405B 39      OUTRTS RTS                 AND RETURN TO CALLER
             *
             *
             * INIT - THE INITIALIZATION ROUTINE
             *
405C 7F 4001 INIT   CLR       INHBIT    ENABLE OUTPUT
405F 7F 4002        CLR       BUFULL    SAY BUFFER EMPTY
4062 01             NOP                 ENABLE INTERRRUPTS
4063 0E             CLI
4064 39             RTS                 RETURN
             *
             *
             * PATCHES TO 680 ASSEMBLY LANGUAGE
             * DEVELOPMENT SYSTEM TO FACILITATE
             * THE USE OF CONTROL CHARACTER
             * ROUTINES - PATCHES ARE FOR VERSION
             * 1.0 ALDS
             *
             * JUMP HERE TO START EDITOR
             *
4065 8D F5   EDIT   BSR       INIT      CALL INIT
4067 7E 0107        JMP       $0107     GO TO EDITOR
             *
             * JUMP HERE TO REENTER EDITOR
             *
406A 8D F0   EDITR  BSR       INIT      CALL INIT
406C 7E 010A        JMP       $010A     REENTER EDITOR
             *
             * JUMP HERE TOO START ASSEMBLER
             *
406F 8D EB   ASM    BSR       INIT      CALL INIT
4071 7E 010E        JMP       $010E     START ASSEMBLER
             *
             *
                    OPT       NOM
             *
             * PATCHES TO ASSEMBLER/EDITOR
             *
0183                ORG       @603
0183 BD 4039        JSR       INCHAR
             *
022D                ORG       @1055
022D BD 404A        JSR       OUTCHR
             *
0175                ORG       @565
0175 91             FCB       $91
             *
0100                ORG       $0100
0100 7E 4004        JMP       INTSRV
             *
             *
             * PATCHES TO STAND ALONE EDITOR
             *
0168                ORG       @550
0168 BD 4039        JSR       INCHAR
             *
01ED                ORG       @755
01ED BD 404A        JSR       OUTCHR
             *
015A                ORG       @532
015A 91             FCB       $91
             *
0100                ORG       $0100
0100 7E 4004        JMP       INTSRV
             *
                    END

TOTAL ERRORS 00000
```

INCHAR waits for BUFULL to go TRUE and reads a character out of CHRBUF. Once the character is read, BUFULL is set to FALSE. The character read is usually passed back to the calling program in one of the processor's registers.

3. OUTCHR - The output character routine (See Figure 4.)

OUTCHR is called to output a character to the terminal. The character to be output is usually passed to OUTCHR in one of the processor's registers. OUTCHR checks the state of the inhibit output flag, INHBIT. This is how control-O takes effect since it controls the state of INHBIT. If INHBIT is TRUE, then OUTCHR simply returns control to the calling program without sending the character to the terminal. If INHBIT is FALSE, then OUTCHR loops, waiting for the terminal output status bit to indicate that the terminal is ready to receive a character. OUTCHR then outputs the character and returns to the calling program.

4. INIT - The initialization routine (See Figure 5.)

8.5

Flowchart for OUTCHR

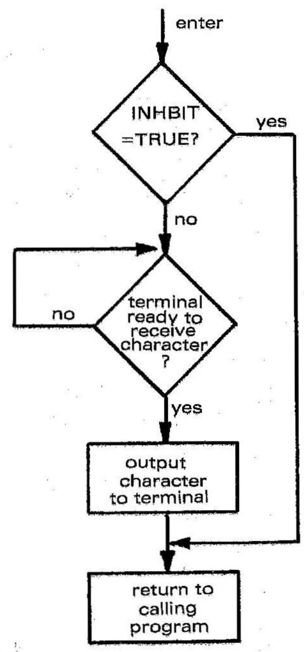


Figure 4

8.6

Flowchart for INIT

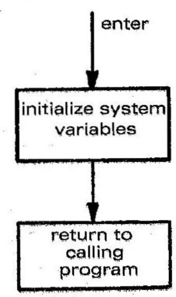


Figure 5

INIT is called to initialize the system variables. For example, INIT might set INHBIT to FALSE and enable keyboard interrupts by initializing the terminal interface accordingly. The exact actions taken by INIT can be tailored to best suit the particular implementation.

A Note on Interrupts

When an interrupt occurs, the state of the machine must be saved so that the interrupted program can later resume where it left off. This is usually handled by pushing

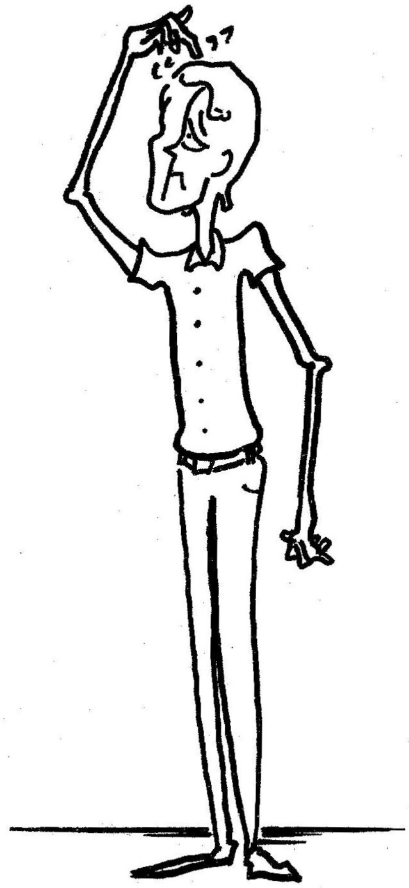
Machines Have Languages?

By Ken Knecht
539 Addison Ave.
Chicago, Ill. 60613

Knecht is chief engineer at the Chicago City College TV Production Center. He's the author of DESIGNING AND MAINTAINING THE SMALL TV AND CATV STUDIO.

My computer education began in 1975 when I decided to buy an Altair microcomputer. I sent away for a MITS sales brochure and had a wonderful time deciding what hardware to get. Then I decided I'd better get some software, whatever that was. (I presumed it had something to do with programming the computer.) That's where I really ran into some semantic difficulties.

The brochure said I had a choice of three different versions of BASIC (4K, 8K and Extended). I noticed that the 8K version had additional trigonometric functions (at least that was a familiar term). But even after



reading the descriptions of each version, I was still confused about computer languages.

The brochure also mentioned strings, double precision Extended BASIC and disk file I/O. That part really left me blank. I didn't have a disc on order and didn't think I needed double precision, so I split the difference and ordered the 8K. If debug meant what I thought it did, I knew that I would need it. So I ordered it too. Fortunately, I didn't ask myself why I was spending all that money for something I knew nothing about.

While waiting for the computer to arrive, I bought a few books on BASIC and finally found out what I was spending my hard-earned money for. My decision to order 8KBASIC had been a good one. The string functions in 8K BASIC allowed me to use words and sentences as variables in the program and manipulate them as I wished.

By looking up some BASIC commands in my books and then checking to see if they were in the 8K version (they usually were), I

PROGRAM CONTROL

Continued

the PC end other registers onto the stack. Although there are exceptions, it is customary to disable interrupts while processing an interrupt. The Motorola 6800 MPU takes care of both problems automatically by pushing all registers onto the stack and disabling interrupts in response to an interrupt request.

Implementation for the Altair 680b

This section describes an implementation of the design for the Altair 680b. Complete assembly listings of the routines as well as patches to integrate the routines into the 680 Assembler and Editor are given. Readers not interested in this material should go to the last section of this article, "Some Further Considerations."

The implementation for the 680b is straightforward because the code follows the flow charts very closely. The following points are related to the code:

1. In response to an interrupt, the 6800 MPU automatically disables interrupts and saves all the registers on the stack.
2. The RTI (return from interrupt) instruction restores all the registers, including the interrupt mask bit in the condition codes register.
3. Due to a "bug" in the 6800 MPU, the CLI (clear interrupt mask) and SEI (set interrupt mask) instructions do not operate properly under all conditions. However, if these instructions are preceded by a NOP instruction, they will perform properly.
4. All the flags used in the code are FALSE when zero and TRUE when non-zero.
5. INTSRV uses indexed addressing for the flags and character buffer to reduce memory usage.
6. INCHAR checks the Monitor's echo flag, ECOFLG, and echoes the character it read if the high order bit of ECOFLG is clear. (See 680b PROM Monitor Manuals for details.)
7. The code has been tested with the 680 Assembler and Editor.

Some Further Considerations

1. There are times when a program would not want control characters to be screened from its input. For example, when a binary paper tape is being loaded through the terminal, a control-C (3) is very likely to appear in the input stream as load data. What modifications would be necessary to handle this situation?
2. INTSRV assumes that the terminal is the interrupting device. In a system with a real time clock that causes interrupts, INTSRV would not perform correctly. What changes would be necessary to make INTSRV work on a system where interrupts are caused by more than one device?
3. (This one is for experts.) What would happen if someone held down the CONTROL, S and REPEAT keys indefinitely? What factors determine how long it would take for this result to occur? How could INTSRV be changed to prevent this problem from occurring?

Feedback

I will be happy to respond to any written comments, suggestions and questions regarding this article. Time does not permit a discussion of these matters over the phone.

got an idea of what I could do with my computer once I built it.

Of course, in those early days of personal computing, there was a big difference between building a computer and getting it to run. Not knowing what I was doing or why I was doing it didn't help either. At that time there were no Altair Computer Centers, books on microcomputing or hobbyist magazines to help me. Even when BYTE finally came out, it was miles over my head. So it was me, the computer and the phone line to MITS.

Getting BASIC and my Altair 8800 up and running was a big battle but a real learning experience. Most of the problems were mine but with some help from MITS, I finally got everything going. Before tackling BASIC, I decided to try some machine language programming.

MACHINE LANGUAGE FIRST

I ran some addition programs in the 8800a Operator's Guide and decided this is one heck of an inefficient way to add numbers. The multiplication program was even worse. With no monitor running I had to flip front panel switches to enter the programs.

Next, I decided to write a memory checking program. Not knowing any better, I thought if I wrote all 1's into each memory location and then read them back, I could find bad memory locations by looking for zeros. With a little help from BUG BOOK III, I finally wrote a program that did just that. This book gave me good insight into just how each 8080 machine language command works.

I also tried a machine language program that prints out the locations of the memory errors. But I never did get it to run.

ON TO BASIC

It was a monumental struggle to get BASIC to run. First, I found that I had a wrong jumper on the I/O board (8 bits still sounds more logical than the 7 bits required). Then I had ACR alignment problems. (I didn't know that some audio cassette players require disassembly to set the tape speed and holding them upside down to get at the speed adjustment. Of course, after they're put back together and set rightside up, the speed changes again.) Finally, I got it all together so that elusive "Memory Size?" would print.

My next problem was to load the boot loader via toggle switch, start the cassette machine and hope no one in my apartment building put a load on the electrical service for four minutes. When that happens, the fan motor in the computer slows down and a string of "C"'s appears on the terminal. I finally broke down and got a Sola voltage regulating transformer and put it in the AC line between the computer and wall plug. That cured the problem.

BACK TO MACHINE LANGUAGE

About this time I got the debug I had ordered. MITS had kindly included the whole Package II (monitor, text editor and assembler) with the debug. I also received a lot of documentation and a whole new type of software for me to try to learn — assembly language programming. To the novice, assembly language programming is like learning a whole new language. Even an understanding of machine language programming doesn't help much in operating the system. There was no book equivalent to BUGBOOK III to go to either. I finally found a book, **COMPUTER PROGRAMMING HANDBOOK,** published by Tab 752, that gave me a little insight into assembly language programming. It was helpful once I got through the long discussions of number systems, how to convert from one to the other, how to multiply in octal, etc. Multiply in octal? Good grief! I'm still looking for a better book on assembly language programming for the microcomputer. There are plenty on the IBM 360 and 370, but they're not much help.

I must admit that I was overwhelmed by the Package II documentation and never really gave it a serious try for any major programs. I had BASIC and couldn't think of any reason to use machine language instead. But I have several ideas for programs using machine language, such as a memory test program and a text editing routine to use with my word processor BASIC program. I've discovered that some things are not only easier to do in machine language, but sometimes that's the only practical way.

For example, I want to be able to emulate the MITS BASIC text editor in my program but can't do it in BASIC because I need to suppress echoing some of my input. There's a control 0 statement that stops printout, but it doesn't work for input statements. So I'll have to do it in machine language, and of course, the assembler makes such programming much easier. Unfortunately, I've got a disc now and hate to go back to cassette tape. So I'll order a MITS DOS. Another example is a memory test program I have in mind. I first saw the MITS memory testing program in firmware run at my local Altair Computer Center. I want to write my own version to put in PROM. While I'm waiting for my DOS, I'll write both programs in assembly language and then do testing and editing when DOS arrives.

Games

I first ran some carefully selected programs from David Ahl's **101 BASIC COMPUTER GAMES.** I based my selection on whether the programs were short enough to load quickly.

I started with the old favorite "Guess," in which the computer selects a number at random and you have to guess what it is. The computer tells you if you're too high or too low. It may not be the most interesting game, but everyone has to start somewhere. After waiting so long to get something running on my computer, even "Guess" was fun for a while.

I tried some more complex games in Ahl's book but found that I couldn't run all of them because they ran poorly or slowly, or I couldn't convert from the DEC BASIC to MITS BASIC. "Hex" is one such game. I still haven't been able to convert it to MITS string functions so it will run properly.

Several months later I began writing my own games. But there was always the awesome challenge of a blank sheet of paper and the marvelous program in my mind. Top down programming? Never heard of it. I started with line 1 and continued from there. I decided that it's easier to skip to the end of the program and add a subroutine to make a previous subroutine work. Even if the routines are a bit mixed up, the program still runs. Since the programs were for my own personal use, I wasn't concerned with legibility.

Of course, this method had its disadvantages—like when I tried to run the program two weeks later and it crashed, giving me the excuse "UNDEFINED LINE IN 113". Hmm, I must have moved a subroutine from that line for some reason. But what subroutine and where was it now? I wished that I hadn't taken the memory-saving step of leaving out the comments. So I went back to the original handwritten program. (Of course I had saved it. I save everything.) But so many lines were scratched out, moved or replaced that I couldn't read it anyway. So I wasted a half hour trying to decipher my code and keep track of all the handy GOTO's I added when things weren't going well.

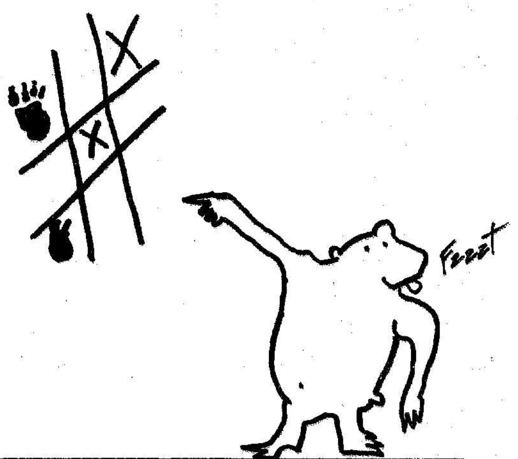
Childhood Wish Fulfilled with TIC-TAC-TOE

By Ken Knecht

Ever since I was a little kid and saw the Bell Telephone TIC-TAC-TOE machine in the Museum of Science and Industry in Chicago, I wanted a machine that would play the game with me. As I grew older and learned about electronics, I tried to figure out how to use relays, transistors and IC's to set up the game. But not until I bought my own microcomputer in 1975 was I able to get my wish.

Although the following routine seems unbeatable, it really isn't very aggressive. It will usually tie but doesn't win very often unless you do something rather stupid. But I'm trying to improve it. I've already learned that a side or corner opening is supposed to be better than the center opening I chose.

The game must be played to the end, because no ending decision line is included. To get out of the program, type "control C"



```
LIST 'TIC TAC TOE

1 PRINT"WANT DIRECTIONS":INPUT A$:IF LEFT$(A$,1)<>"Y" THEN 57
10 PRINT"PLAY TIC TAC TOE WITH THE COMPUTER."
20 PRINT"YOU MOVE FIRST. TO WIN PLAYER MUST"
30 PRINT"FILL ROW, COLUMN OR DIAGONAL. IF NEITHER"
40 PRINT"CAN THEN A TIE IS CALLED. "
51 PRINT:PRINT"YOU MOVE FIRST, TYPE IN THE R (ROW)"
52 PRINT"NUMBER AFTER THE FIRST QUESTION MARK,"
53 PRINT"THE C (COLUMN) NUMBER AFTER THE SECOND."
54 PRINT"SEE THE FOLLOWING DIAGRAM. BOARD WILL BE"
55 PRINT"REPEATED AFTER EVERY COMPUTER MOVE."
57 DIM C(3,3),P(3,3),C$(3,3)
60 BL=0:FOR X=1 TO 3
70 FOR Y=1 TO 3
80 C(X,Y)=0:P(X,Y)=0:C$(X,Y)="*"
90 NEXT Y
100 NEXT X
160 PRINT:PRINT:PRINT
170 GOSUB 500 'PRINT BOARD
175 PRINT
180 GOSUB 630 'PLAYER'S MOVE
190 IF C(2,2)=0 THEN C(2,2)=3:C$(2,2)="C":GOTO 201
200 IF C(2,2)>0 THEN C(1,3)=3:C$(1,3)="C"
201 GOSUB 500
202 GOSUB 630
203 IF C(2,2)=3 AND C(1,1)=1 AND C(3,3)=1 THEN C(1,2)=3:C$(1,2)="C":
GOTO 210
204 IF C(2,2)=3 AND C(1,3)=1 AND C(3,1)=1 THEN C(1,2)=3:C$(1,2)="C":
GOTO 210
205 IF C(2,2)=1 AND C(3,1)=1 AND C(1,3)=3 AND C(1,1)=0 THEN
C(1,1)=3:C$(1,1)="C":GOTO 210
206 IF C(2,2)=1 AND C(1,1)=1 AND C(3,3)=3 AND C(3,1)=0 THEN
C(3,1)=3:C$(3,1)="C":GOTO 210
209 GOTO 230
210 GOSUB 500 'PRINT BOARD
220 GOSUB 630 'PLAYER'S MOVE
230 GOSUB 710 'CHECK FOR 2 IN A ROW
240 GOSUB 880 'CHECK FOR 2 IN A COLUMN
250 GOSUB 940 'CHECK FOR 2 IN A DIAGONAL
260 GOSUB 1050 'CHECK FOR FULL ARRAY
270 IF BL=1 THEN BL=2:GOTO 230
280 IF C(2,2)=3 AND C(1,2)=1 AND C(1,1)=0 AND C(1,3)=0 THEN
C(1,1)=3:C$(1,1)="C":GOTO 210
290 IF C(2,2)=3 AND C(2,3)=1 AND C(1,3)=0 AND C(3,3)=0 THEN
C(1,3)=3:C$(1,3)="C":GOTO 210
```

Machines Have Languages?

Continued

Oh well, some people have to learn the hard way. I've been programming for over a year and still astonish myself with the dumb mistakes I continue to make. For example, earlier this evening I decided to put the documentation for a program I hope to sell in a file on disc so I can print copies whenever I want. So I wrote a simple five-line program to record the stuff on disc. It worked very well except for one slight flaw - I forgot to put in a line to print the input line on disc. I'm sure the computer wondered what I was doing typing in all those sentences and then throwing them away and replacing them — again and again and again. Two hours of typing documentation waste. Next time I'll have to remember to try the program first.

Here's another mistake I recently made. How about FOR X%= .1 to .9 STEP .1? That doesn't work very well, because the % forces the argument to integer. So I had FOR X% = 0 to 0 STEP.1. Not too effective. But it sure sounded great when I wrote it.

I'm trying to use structured programming now, and some of the shorter programs even end up that way. But I still fall prey to the temptation of fixing problems quickly with a handy GOTO or GOSUB to a vacant line somewhere when I need to fix a routine. Bad habits die slowly, but I'm trying.

or add 870 and 1160. A "no player wins" line is also included for when the game ends in a tie. The program is primarily predetermined moves based on previous moves. It does its own blocking and line completion but doesn't make any intricate determination as to what the next move should be.

It would be interesting to come up with an algorithm for deciding where to move. One possibility is to number the squares in the TIC-TAC-TOE field like a magic square. For example, row l;2,9,4: row 2; 7,5,3: row 3;6,l,8. The sum of each column, row or diagonal equals 15. The computer must then try to get three numbers totaling 15 before the other player while blocking the opponent's attempt to do so.

The program could be written to be self-teaching—the computer puts all nine positions in a string at each move together with the move it made from the current position. If the computer loses, it would erase or disregard the last move and try another when that position reappeared. If all tries from that position result in a losing game, it would move back to a previous position and try all the possibilities there.

The following program contains enough comments so that any modifications can be easily added.

```
300 IF C(2,2)=3 AND C(2,1)=1 AND C(1,1)=0 AND C(3,1)=0 THEN
C(1,1)=3:C$(1,1)="C":GOTO 210
310 IF C(2,2)=3 AND C(3,2)=1 AND C(3,1)=0 AND C(3,3)=0 THEN
C(3,3)=3:C$(3,3)="C":GOTO 210
370 IF C(2,1)=1 AND C(1,3)=1 AND C(2,2)=3 AND C(1,1)=0 THEN C(1,1)=3
:C$(1,1)="C":GOTO 210
380 IF C(1,2)=1 AND C(3,3)=1 AND C(2,2)=3 AND C(1,3)=0 THEN C(1,3)=3:
C$(1,3)="C":GOTO 210
390 IF C(2,3)=1 AND C(3,1)=1 AND C(2,2)=3 AND C(3,3)=0 THEN C(3,3)=3:
C$(3,3)="C":GOTO 210
400 IF C(1,1)=1 AND C(3,2)=1 AND C(2,2)=3 AND C(3,1)=0 THEN C(3,1)=3:
C$(3,1)="C":GOTO 210
410 IF (X=1 AND Y=2 AND C(1,3)=0) OR (X=2 AND Y=3 AND C(1,3)=0) THEN
C(1,3)=3:C$(1,3)="C":GOTO 210
420 IF (X=2 AND Y=3 AND C(3,3)=0) OR (X=3 AND Y=2 AND C(3,3)=0) THEN
C(3,3)=3:C$(3,3)="C":GOTO 210
430 IF (X=3 AND Y=2 AND C(3,1)=0) OR (X=2 AND Y=1 AND C(3,1)=0) THEN
C(3,1)=3:C$(3,1)="C":GOTO 210
440 IF (X=2 AND Y=1 AND C(1,1)=0) OR (X=1 AND Y=2 AND C(1,1)=0) THEN
C(1,1)=3:C$(1,1)="C":GOTO 210
450 IF (X=1 AND Y=1 AND C(1,2)=0) OR (X=1 AND Y=3 AND C(1,2)=0) THEN
C(1,2)=3:C$(1,2)="C":GOTO 210
460 IF (X=1 AND Y=3 AND C(2,3)=0) OR (X=3 AND Y=3 AND C(2,3)=0) THEN
C(2,3)=3:C$(2,3)="C":GOTO 210
470 IF (X=3 AND Y=3 AND C(3,2)=0) OR (X=3 AND Y=1 AND C(3,2)=0) THEN
C(3,2)=3:C$(3,2)="C":GOTO 210
480 IF (X=3 AND Y=1 AND C(2,1)=0) OR (X=1 AND Y=1 AND C(2,1)=0) THEN
C(2,1)=3:C$(2,1)="C":GOTO 210
500 PRINT:PRINT:PRINT
530 PRINT TAB(5);"C1";TAB(9);"C2";TAB(13);"C3"
540 PRINT TAB(2);"R1";TAB(5);C$(1,1);TAB(7);"I";TAB(9);C$(1,2);TAB(11);
"I";TAB(13);C$(1,3)
550 PRINT TAB(4);"-----------"
560 PRINTTAB(2);"R2";TAB(5);C$(2,1);TAB(7);"I";TAB(9);C$(2,2);TAB(11);
"I";TAB(13);C$(2,3)
570 PRINT TAB(4);"-----------"
580 PRINT TAB(2);"R3";TAB(5);C$(3,1);TAB(7);"I";TAB(9);C$(3,2);TAB(11);
"I";TAB(13);C$(3,3)
620 RETURN
630 PRINT"YOUR MOVE. ROW?":INPUT X
640 IF X>3 OR X<1 THEN 1180
650 PRINT"YOUR MOVE. COLUMN?":INPUT Y
660 IF Y>3 OR Y<1 THEN 1180
670 IF C(X,Y)>0 THEN 700
680 C(X,Y)=1:C$(X,Y)="P"
690 RETURN
700 PRINT"THAT SPACE IS ALREADY TAKEN: TRY AGAIN.":GOTO 630
710 S=0:C=0:FOR A=1 TO 3
720 FOR B=1 TO 3
730 GOSUB 770 'CHECK FOR 2 OR 3 IN A ROW
740 NEXT B
750 S=0:C=0:NEXT A
760 RETURN
770 IF C(A,B)=0 THEN 850
780 IF C(A,B)>0 THEN S=S+C(A,B)
790 IF S=6 AND C=5 THEN 830 'FILL LINE AND WIN
800 IF S=2 AND C=5 THEN GOSUB 1200 'BLOCK AVAILABLE
810 IF S=9 THEN 860
820 RETURN
830 C(A1,B1)=3:C$(A1,B1)="C":GOTO 860
840 C(A1,B1)=3:C$(A1,B1)="C":BL=0:GOTO 210
850 A1=A:B1=B:C=5:GOTO 790
860 GOSUB 500
```

Program Continued on Following Page

Program Useful for Number Conversion

by Pat Diettmann

This program first appeared in the *AMATEUR* COMPUTER GROUP OF NEW JERSEY NEWS, Vol. 3, No. 6, June 1977. It will run in MITS 8K BASIC, version 4.0.

```
LIST

2 REM BINARY BASIC BY M.G. 'PAT' DIETTMANN
5 PRINT "NUMERIC SYSTEM CONVERSIONS"
10 PRINT "DECIMAL","BINARY","","HEX","OCTAL"
15 LET C=A
20 INPUT"WHICH SYSTEM(D,B,H,O)    ";A$
30 IF A$="D" THEN 400
40 IF A$="B" THEN 300
50 IF A$="H" THEN 100
60 IF A$="O" THEN 200
70 GOTO 20
100 INPUT"HEX      ";K$
102 K$="0000"+K$:K$=RIGHT$(K$,4)
105 L$="":REM NULL L$
110 FOR J=1TO4
120 J$=MID$(K$,J,1)
130 GOSUB 2400
140 L$=L$+E$:NEXT J
145 A$=L$:GOSUB 1500:GOSUB 2200
150 PRINTA,L$,K$,C$
160 GOTO 100
200 INPUT"OCTAL   ";K$
205 K$="000000"+K$:K$=RIGHT$(K$,6)
210 L$="":REM NULL L$
220 FOR J=1TO6
230 J$=MID$(K$,J,1)
240 GOSUB 2400
250 IF LEFT$(E$,1)="1" THEN E$="****"
260 L$=L$+RIGHT$(E$,3)
270 NEXT J:A$=RIGHT$(L$,16):L$=A$
280 GOSUB 1500:GOSUB 2300
290 PRINT A,L$,B$,K$:GOTO 200
300 INPUT"BINARY";A$
310 A$="0000000000000000"+A$:A$=RIGHT$(A$,16)
320 L$=A$:GOSUB 1500
330 GOSUB 2200
340 GOSUB 2300
350 PRINT A,A$,B$,C$
360 GOTO 300
400 INPUT"DECIMAL";A
410 LET C=A
420 GOSUB 1000:GOSUB 2300:GOSUB 2200
430 PRINT C,A$,B$,C$
440 GOTO 400
1000 REM CONVERT DECIMAL(A) TO BIN(A$) 16 BIT
1005 LET B=65536
1010 IF A>(B-1) THEN A$="**ERROR**":RETURN
1020 LET A$="":REM NULL A$
1030 FOR I=1TO16
1040 B=INT(B/2+.2)
1050 A=A-B
1060 IF A<0 THEN A=A+B:A$=A$+"0":GOTO 1080
1070 A$=A$+"1"
1080 NEXT:RETURN
1500 REM CONVERT BIN 16 BIT(L$) TO DECIMAL(A)
1510 A=0:B=32768
1520 FOR I=1TO16
1530 IF MID$(L$,I,1)="0" THEN 1550
1540 A=A+B
1550 B=INT(B/2+.2):NEXT I:RETURN
```

Program Continued on Following Page

Program Useful for Number Conversion

Continued

```
2000 DATA '0000','0','0001','1','0010','2','0011','3'
2010 DATA '0100','4','0101','5','0110','6','0111','7'
2020 DATA '1000','8','1001','9','1010','A','1011','B'
2030 DATA '1100','C','1101','D','1110','E','1111','F'
2040 DATA 'K2PPZ','4/8/77'
2100 REM CONVERT 4 BIT BIN(D$) TO HEX DIGIT(F$)
2110 RESTORE
2120 FOR I=1TO16
2130 READ E$,F$
2140 IF D$=E$ THEN RETURN
2150 NEXT:F$='*':RETURN
2200 REM CONVERT BIN(A$)TO OCTAL(C$)
2210 C$='':REM NULL C$
2220 G$='00'+A$
2230 FOR J=1TO16 STEP 3
2240 D$=MID$(G$,J,3):D$='0'+D$
2250 GOSUB 2100
2260 C$=C$+F$:NEXT J
2270 RETURN
2300 REM CONVERT 16 BIT BIN(A$) TO HEX(B$)
2310 B$='': REM NULL B$
2320 FOR J=1TO13 STEP 4
2330 D$=MID$(A$,J,4)
2340 GOSUB 2100
2350 B$=B$+F$:NEXT J
2360 RETURN
2400 REM CONVERT HEX DIGIT(J$) TO 4 BIT BIN(E$)
2410 RESTORE
2420 FOR I=1TO16
2430 READ E$,F$
2440 IF J$=F$ THEN RETURN
2450 NEXT:E$='****':RETURN
3000 REM TO ELIMINATE DATA REQUIREMENT
3010 REM USE THE FOLLOWING
3015 REM  FOR BIN TO HEX OR OCTAL
3020 IF D$='0000' THEN F$='0':RETURN
3030 REM 0001 TO1110 USE 1 TO E
3170 IF D$='1111' THEN F$='F':RETURN
3180 F$='*':RETURN
3200 REM  FOR HEX TO BIN
3210 IF J$='0' THEN E$='0000':RETURN
3220 REM 1 TO E USE 0001 TO 1110
3360 IF J$='F' THEN E$='1111':RETURN
3370 E$='****':RETURN
OK


1535 IF MID$(L$,I,1)><'1' THEN A=-1E5:GOTO 1550
RUN
NUMERIC SYSTEM CONVERSIONS
DECIMAL        BINARY                        HEX           OCTAL
WHICH SYSTEM(D,B,H,O)    ? H
HEX     ? ABXC
-99988         10101011****1100              ABXC          125**4
HEX      ?

OK

RUN
NUMERIC SYSTEM CONVERSIONS
DECIMAL        BINARY                        HEX           OCTAL
WHICH SYSTEM(D,B,H,O)    ? D
DECIMAL? 0
 0             0000000000000000              0000          000000
DECIMAL? 65535
 65535         1111111111111111              FFFF          177777
DECIMAL? 65536
 65536         **ERROR**       ****          ******
DECIMAL? 32767
 32767         0111111111111111              7FFF          077777
DECIMAL? 32768
 32768         1000000000000000              8000          100000
DECIMAL? 4096
 4096          0001000000000000              1000          010000
DECIMAL? 4095
 4095          0000111111111111              0FFF          007777
DECIMAL? 256
 256           0000000100000000              0100          000400
```

Program continued on following page

Childhood Wish Fulfilled with TIC-TAC-TOE

Continued

```
870 PRINT"COMPUTER WINS. TRY AGAIN?":GOTO 60
880 S=0:C=0:FOR B=1 TO 3
890 FOR A=1 TO 3
900 GOSUB 770 'CHECK FOR 2 OR 3 ALIKE IN A COLUMN
910 NEXT A
920 S=0:C=0:NEXT B
930 RETURN
940 S=0:C=0
950 A=1:B=1:GOSUB 770
960 A=2:B=2:GOSUB 770
970 A=3:B=3:GOSUB 770
980 'CHECK FOR 2 OR 3 IN A DIAGONAL
990 S=0:C=0
1000 A=1:B=3:GOSUB 770
1010 A=2:B=2:GOSUB 770
1020 A=3:B=1:GOSUB 770
1030 'CHECK FOR 2 OR 3 IN A DIAGONAL
1040 RETURN
1050 S=0:F=0:FOR A=1 TO 3
1060 FOR B=1 TO 3
1070 GOSUB 1110 'CHECK FOR FULL ARRAY
1080 NEXT B
1090 S=0:NEXT A
1100 RETURN
1110 IF C(A,B)>0 THEN S=S+C(A,B)
1120 IF S=5 OR S=7 THEN 1140 'LINE IS FULL
1130 RETURN
1140 F=F+1
1150 IF F<3 THEN 1130
1160 IF F=3 THEN PRINT"GAME IS TIED. TRY AGAIN?"
1170 GOTO 60
1180 PRINT"ILLEGAL MOVE. DO NOT TYPE IN A NUMBER GREATER"
1190 PRINT"THAN 3 OR LESS THAN 1.":GOTO 630
1200 IF BL=2 THEN 840
1210 BL=1:RETURN
1220 GOSUB 500
1230 PRINT:PRINT:PRINT"PLAYER WINS. TRY AGAIN?":GOTO 60
OK
```

```
OK
RUN
NUMERIC SYSTEM CONVERSIONS
DECIMAL          BINARY                      HEX           OCTAL
WHICH SYSTEM(D,B,H,O)    ? B
BINARY? 0
 0               0000000000000000            0000          000000
BINARY? 1010101010101010
 43690           1010101010101010            AAAA          125252
BINARY? 01010101010101010101
 21845           0101010101010101            5555          052525
BINARY? 1
 1               0000000000000001            0001          000001
BINARY? 11111111
 255             0000000011111111            00FF          000377
BINARY? 1111111100000000
 65280           1111111100000000            FF00          177400
BINARY? 1000000000000000
 32768           1000000000000000            8000          100000
BINARY? 0100000000000000
 16384           0100000000000000            4000          040000
BINARY? 123456789
 511             0000000123456789          01**        * 000***
BINARY? 99
 3               0000000000000099          000*        * 00000*
BINARY?

OK
RUN
NUMERIC SYSTEM CONVERSIONS
DECIMAL          BINARY                    HEX           OCTAL
WHICH SYSTEM(D,B,H,O)    ? H
HEX      ? 0
 0               0000000000000000          0000          000000
HEX      ? FFFF
 65535           1111111111111111          FFFF          177777
HEX      ? 8000
 32768           1000000000000000          8000          100000
HEX      ? 4000
 16384           0100000000000000          4000          040000
HEX      ? 2000
 8192            0010000000000000          2000          020000
HEX      ? 1000
 4096            0001000000000000          1000          010000
HEX      ? 0800
 2048            0000100000000000          0800          004000
HEX      ? 400
 1024            0000010000000000          0400          002000
HEX      ? 200
 512             0000001000000000          0200          001000
HEX      ? 100
 256             0000000100000000          0100          000400
HEX      ? 0080
 128             0000000010000000          0080          000200
HEX      ? 040
 64              0000000001000000          0040          000100
HEX      ? 20
 32              0000000000100000          0020          000040
HEX      ? 10
 16              0000000000010000          0010          000020
HEX      ? 8
 8               0000000000001000          0008          000010
HEX      ? 4
 4               0000000000000100          0004          000004
HEX      ? 2
 2               0000000000000010          0002          000002
HEX      ? 1
 1               0000000000000001          0001          000001
HEX      ? FF00
 65280           1111111100000000          FF00          177400
HEX      ? 00FF
 255             0000000011111111          00FF          000377
HEX      ? F0F0
 61680           1111000011110000          F0F0          170360
HEX      ? WXYZ
 65535           ****************          WXYZ        * ******
HEX      ? XABCCDEF
 52719           1100110111101111          CDEF        * 146757
HEX      ? XABC
 64188           ****101010111100          XABC          **5274
HEX      ? ABXC
 44028           10101011****1100          ABXC          125**4
HEX      ?
```

Program continued
on following page

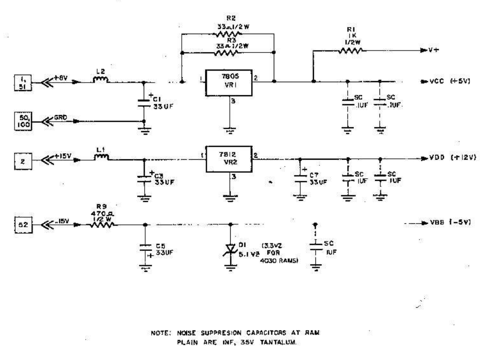
* Indicates Note 1 in text.

Altair 88-S4K Power Supply Schematic

By Glenn Wolf

The following errors should be noted on the Altair 88-S4K board. The J & K inputs to the second half of Flip Flop E are tied to V+ *not* Vcc. The Power Supply schematic was also never published with the manual.

Notice in the schematic that the purpose of R2 & R3 are to relieve some of the load from the regulator while still maintaining a regulated output voltage. The purpose of R1 is to provide I limiting for the V+t1 line, which goes to inputs of chips that are unprotected internally. The VDD (t1l2v) supply goes to pin l8 of the RAMS and is also used for the generation of the RAM chip Enable pulse, which is supplied to pin l7 of the RAMS. The VBB (-5v) supply is regulated with a 5.lv zener. Depending on the type of RAMS used, this zener may have to be changed. For example, 4030 RAMS would require a 3.3v zener.



Command Changed

The Altair 680 BASIC command for outputting to the KCACR control channel should be changed from POKE-4080 to POKE-61456. (See pp. 30 and 3l, paragraphs 2-l5 and 2-l9.) The negative complement of the address is no longer used. When using PEEK or POKE in Altair 680 CSAVE BASIC, the actual decimal address may be used.

HARDWARE

Using Sector Interrupts on the Altair Floppy

By Thomas Durston

Although the interrupt circuit of the Floppy Disk Controller is an extremely useful feature, not many users take advantage of it because it is not utilitzed in Altair Disk BASIC. The primary functions of the interrupt procedures listed here are to save CPU time in systems that are supporting Floppy Disks and other operations simultaneously.

The most obvious use of the sector interrupts is to perform a sector search for disk I/O. The disk controller interrupts the Altair computer at the beginning of every sector, or every 5.2 ms. Only a few microseconds are used to identify the sector count, and the Altair computer can perform other tasks until the required sector is found.

Interrupts can also be used for disk system timing. Instead of checking Move Head status every time the disk head is stepped, the timing can be controlled by sector interrupts. This is done by enabling interrupts and issuing the desired step command after the first interrupt is received. Then, since interrupts occur every 5.2 ms, count two interrupts (10.4ms), and issue the next step command. This process may be used to step the disk head any number of tracks (within 0 to 76) without checking MH status. When looking for track Ø, check for trackØ status before issuing the step command. If you change step direction, wait a minimum of 30 ms or 6 sector interrupts between opposite direction step commands. The only time this change of step direction appears is when referencing to track Ø and immediately stepping to to another track. It usually doesn't occur during reading and writing because of the 45 ms Head Settle delay.

The Head Settle delay may also be simulated by counting sector interrupts. This timing function can be controlled by enabling interrupts, issuing the head load or last step command on the first interrupt and counting 9 more sector interrupts. At the ninth interrupt, the head will be stable for reading or writing data, and the search for the desired sector may begin.

As for the hardware, there are two areas on the controller boards that must be checked before utilizing interrupts. On controller board 2, Rev. 0-X2, there is a track from IC JI, pin 1 which passes by and makes connection to a pad used by C15. This track is on the back side of the P.C. board and must be disconnected from the C15 pad by cutting the land between the track and the pad.

On disk controller board 1, the SRI jumper must be connected to the desired interrupt pad. For single-level interrupts use the INT (or PINT) pad. For Vectored Interrupts, use the highest priority level, VIO. If the board 1 is Rev. 0-X3, solder the jumper wire to the end of the gold finger on the stab connector on the P.C. board.

Program Useful for Number Conversion

Continued

```
OK
RUN
NUMERIC SYSTEM CONVERSIONS
DECIMAL         BINARY                    HEX         OCTAL
WHICH SYSTEM(D,B,H,O)    ? O
OCTAL   ? 0
 0              0000000000000000          0000        000000
OCTAL   ? 177777
 65535          1111111111111111          FFFF        177777
OCTAL   ? 700000
 32768          1000000000000000          8000        700000
OCTAL   ? 100000
 32768          1000000000000000          8000        100000
OCTAL   ? 070000
 28672          0111000000000000          7000        070000
OCTAL   ? 040000
 16384          0100000000000000          4000        040000
OCTAL   ? 123456789
 24063          0101110111******          5D**      * 456789
OCTAL   ? 125252
 43690          1010101010101010          AAAA        125252
OCTAL   ? 052525
 21845          0101010101010101          5555        052525
OCTAL   ?

OK
```

* Indicates Note 1 in text.

A Definition of Terms:

sub-scribe /ˌsəb-ˈscrib/ *vb* **sub-scribed; sub-scrib-ing** [**ME** *subscriber*] **1:** to sign one's name to a document (as a coupon; as the one below) **2:** to enter one's name for a publication (as **CN-**Computer Notes; one year for **$5.00/ $20.00** per year overseas) **3:** to feel favorably disposed **syn** ASSENT **ant** boggle—**sub-scrib-er** *n*

computer notes

mits a subsidiary of **Pertec Computer Corporation**
2450 Alamo S.E.
Albuquerque, New Mexico 87106

Please send me a 1 year subscription to **Computer Notes.**
$5.00 per year in U.S. $20.00 per year overseas.

NAME:__________

ADDRESS:__________

CITY:__________ STATE:__________ ZIP__________

COMPANY/ORGANIZATION__________

☐ Check Enclosed MC or BAC/Visa #__________
☐ Master Charge Exp Date __________
☐ BankAmericard/Visa Signature__________